(2)

TECHNICAL REPORT RD-GC-92-31

AD-A252 833

REAL TIME EXECUTIVE FOR MISSILE SYSTEMS MC68020 ASSEMBLY INTERFACE

Phillip R. Acuff and
Wanda M. Hughes
Guidance and Control Directorate
Research, Development, and Engineering Center

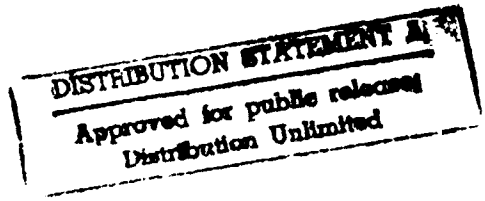
and

On-line Applications Research Corporation
3315 Memorial Parkway SW
Huntsville, AL 35801

MAY 1992

U.S. ARMY MISSILE COMMAND
Redstone Arsenal, Alabama 35898-5000

Approved for Public Release.



92-16546

92 6 23 067

DESTRUCTION NOTICE

FOR CLASSIFIED DOCUMENTS, FOLLOW THE PROCEDURES IN DoD 5200.22-M, INDUSTRIAL SECURITY MANUAL, SECTION II-19 OR DoD 5200.1-R, INFORMATION SECURITY PROGRAM REGULATION, CHAPTER IX. FOR UNCLASSIFIED, LIMITED DOCUMENTS, DESTROY BY ANY METHOD THAT WILL PREVENT DISCLOSURE OF CONTENTS OR RECONSTRUCTION OF THE DOCUMENT.

DISCLAIMER

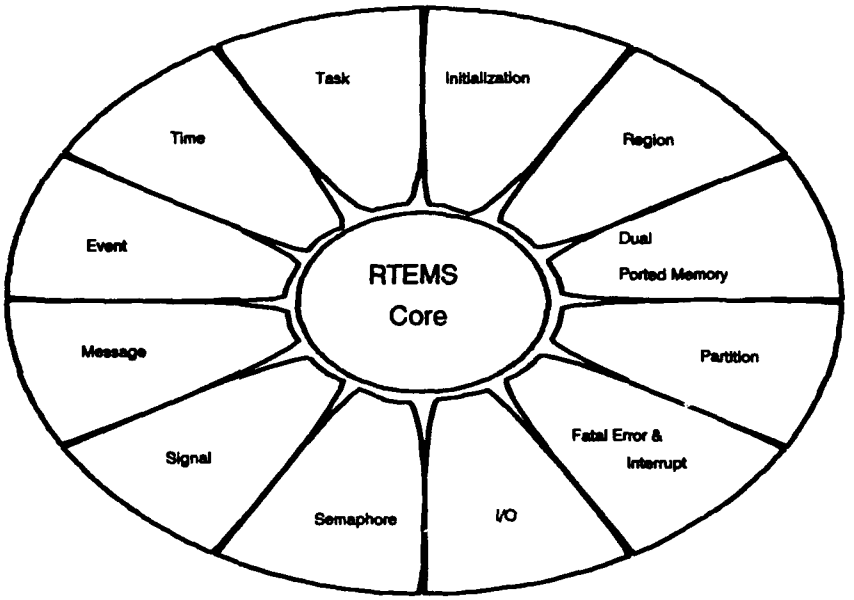
THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

TRADE NAMES

USE OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL ENDORSEMENT OR APPROVAL OF THE USE OF SUCH COMMERCIAL HARDWARE OR SOFTWARE.

Real Time Executive for Missile Systems

MC68020 Assembly Interface



U.S. ARMY MISSILE COMMAND
Redstone Arsenal, Alabama 35898-5254

Release 1.31
December 1991

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) Technical Report RD-GC-92-31	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Guidance & Control Directorate RD&E Center 6b. OFFICE SYMBOL (*If applicable*) AMSMI-RD-GC-S	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (*City, State, and ZIP Code*) Commander, U.S. Army Missile Command ATTN: AMSMI-RD-GC-S Redstone Arsenal, AL 35898-5254	7b. ADDRESS (*City, State, and ZIP Code*)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (*Include Security Classification*) Real Time Executive For Missile Systems MC68020 Assembly Interface	
12. PERSONAL AUTHOR(S) Phillip R. Acuff, Wanda M. Hughes, and OAR Corp.	
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM 6/89 TO 2/92
14. DATE OF REPORT (*Year, Month, Day*) 1992, May	15. PAGE COUNT 79
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
RTEMS, real-time, executive, heterogeneous, homogeneous, multiprocessing, 68020, microprocessor, assembly language,
(Continued on page ii)

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This document details the assembly language interface for the RTEMS (Real-Time Executive for Missile Systems) real-time executive for the Motorola 68020. Each entry in this manual corresponds to a directive in RTEMS. Each directive entry details which registers are used for input arguments and return values in addition to giving an example usage. The assembly language used in the examples is standard Motorola 68020 assembly.

RTEMS is a real-time executive (kernel) which provides a high performance environment for embedded military applications including such features as multitasking capabilities; homogeneous and heterogeneous multiprocessor systems; event-driven, priority-based, pre-emptive scheduling; intertask communication and synchronization; responsive interrupt management; dynamic memory allocation; and a high level of user configurability. RTEMS was originally developed in an effort to eliminate many of the major drawbacks of the Ada

(Continued on page ii)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Wanda M. Hughes	22b. TELEPHONE (*Include Area Code*) (205) 876-4484 22c. OFFICE SYMBOL AMSMI-RD-GC-S

DD Form 1473, JUN 86 *Previous editions are obsolete.*

BLOCK 18 (Cont'd): runtime, directive, multitasking, event-driven, priority-based, preemptive, scheduling, intertask communication, synchronization, dynamic memory allocation, user configurable, kernel, embedded, semaphore, events, interrupt, regions, segments, I/O, messages, user extendable, object oriented

BLOCK 19 (Cont'd): programming language. RTEMS provides full capabilities for management of tasks, interrupts, time, and multiple processors in addition to those features typical of generic operating systems. The code is Government owned, so no licensing fees are necessary. The executive is written using the 'C' programming language with a small amount of assembly language code. The code was developed as a linkable and/or ROMable library with the Ada programming language. Initially RTEMS was developed for the Motorola 68000 family of processors. It has since been ported to the Intel 80386 and 80960 families. This manual describes the assembly language interface to RTEMS for the MC68020 microprocessor. Related documents include: Real Time Executive for Missile Systems User's Guide MC68020 'C' Interface, Real Time Executive for Missile Systems MC68020 Timing Document, and Real Time Executive for Missile Systems MC68020 Ada Interface. RTEMS documentation and code is available for the Motorola 68000 family, and the Intel 80386 and 80960 family of processors.

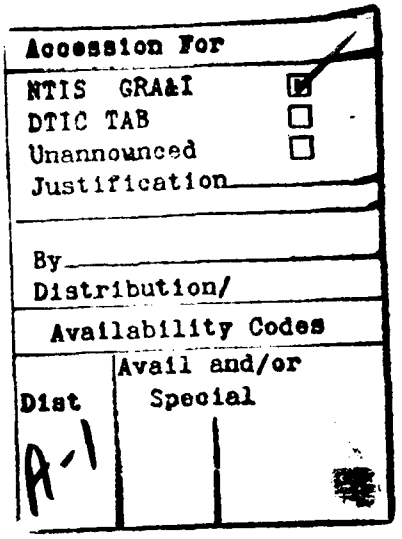


Table of Contents

S.1 Introduction

S.1.1 Description

This supplemental document contains the assembly language interface for the RTEMS real-time executive for the Motorola 68020. For more detailed information regarding exact operation, constants, arguments, and data structures, please refer to the manual page for the appropriate directive.

Each entry in this document corresponds to a directive and details which registers are used for input arguments and return values in addition to an example usage. The assembly language used in the examples is standard Motorola 68020 assembly.

S.1.2 Register Usage

RTEMS-68020 uses the 68020 D0, D1, A0, and A1 registers as scratch registers. The contents of these four registers will not be preserved by RTEMS directives unless noted otherwise.

S.2 INITIALIZATION MANAGER

S.2.1 INIT_EXEC - Initialize RTEMS

INPUT:

D0 = function code
4(SP) = address of configuration table

OUTPUT:

NONE

EXAMPLE:

```
        .
        .
        .

move.l   #Config_tbl,-(SP)     * push address of config table
move.l   #INIT_EXEC,D0         * D0 = function code
jsr      rtems                 * enter the executive

* does not return

        .
        .
        .
```

NOTES:

This directive does not return to the caller.

S.3 TASK MANAGER

S.3.1 T_CREATE - Create a task

INPUT:

D0 = function code
4(SP) = user-defined four byte name
8(SP) = priority
12(SP) = stack size (in bytes)
16(SP) = initial mode
20(SP) = attributes
24(SP) = address of task id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Task_id,-(SP)          * push pointer to task id
move.l   #TASK_ATTRIBUTES,-(SP)  * push attributes
move.l   #TASK_MODE,-(SP)        * push mode
move.l   #STACK_SIZE,-(SP)       * push stack size
move.l   #PRIORITY,-(SP)         * push priority
move.l   #TASK_NAME,-(SP)        * push name
move.l   #T_CREATE,D0            * D0 = function code
jsr      rtems                   * enter the executive

* should check return code here

        .
        .
        .
```

S.3.2 T_IDENT - Get ID of a task

INPUT:

D0 = function code
4(SP) = user-defined name to search for
8(SP) = node identifier (defines search space)
12(SP) = address of task id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Task_id,-(SP)          * push pointer to task id
move.l   #NODE,-(SP)             * push node identifier
move.l   #TASK_NAME,-(SP)        * push task name
move.l   #T_IDENT,D0             * D0 = function code
jsr      rtems                   * enter the executive

* should check return code here

        .
        .
        .
```

S.3.3 T_START - Start a task

INPUT:

D0 = function code
4(SP) = task id
8(SP) = entry point
12(SP) = start argument

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

      move.l   #START_ARG,-(SP)        * push start argument
      move.l   #User_task,-(SP)        * push entry point
      move.l   Task_id,-(SP)           * push task id
      move.l   #T_START,D0             * D0 = function code
      jsr      rtems                   * enter the executive

      * should check return code here

          .
          .
          .
```

S.3.4 T_RESTART - Restart a task

INPUT:

D0 = function code
4(SP) = task id
8(SP) = restart argument

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #RESTART_ARG,-(SP)      * push restart argument
     move.l   Task_id,-(SP)           * push task id
     move.l   #T_RESTART,D0           * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.3.5 T_DELETE - Delete a task

INPUT:

D0 = function code
4(SP) = task id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   Task_id,-(SP)          * push task id
     move.l   #T_DELETE,D0           * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.3.6 T_SUSPEND - Suspend a task

INPUT:

D0 = function code
4(SP) = task id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   Task_id,-(SP)          * push task id
     move.l   #T_SUSPEND,D0          * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.3.7 T_RESUME - Resume a task

INPUT:

D0 = function code
4(SP) = task id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   Task_id,-(SP)          * push task id
move.l   #T_RESUME,D0           * D0 = function code
jsr      rtems                  * enter the executive

* should check return code here

        .
        .
        .
```

S.3.8 T_SETPRI - Set task priority

INPUT:

D0 = function code
4(SP) = task id
8(SP) = new priority
12(SP) = address of previous priority storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #Prev_priority,-(SP)    * push pointer to previous
priority
move.l    #PRIORITY,-(SP)         * push new task priority
move.l    Task_id,-(SP)           * push task id
move.l    #T_SETPRI,D0            * D0 = function code
jsr       rtems                   * enter the executive

* should check return code here

          .
          .
          .
```

S.3.9 T_MODE - Change current task's mode

INPUT:

D0 = function code
4(SP) = new mode
8(SP) = mask
12(SP) = address of previous mode storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Prev_mode,-(SP)        * push pointer to previous mode
     move.l   #MASK,-(SP)             * push mask
     move.l   #MODE,-(SP)             * push new mode
     move.l   #T_MODE,D0              * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.3.10 T_GETNOTE - Get task notepad entry

INPUT:

D0 = function code
4(SP) = task id
8(SP) = notepad entry number
12(SP) = address of note value storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

    move.l   #Note_val,-(SP)        * push pointer to note value
    move.l   #NOTE_NUM,-(SP)        * push entry number
    move.l   Task_id,-(SP)          * push task id
    move.l   #T_GETNOTE,D0          * D0 = function code
    jsr      rtems                  * enter the executive

    * should check return code here
        .
        .
        .
```

S.3.11 T_SETNOTE - Set task notepad entry

INPUT:

D0 = function code
4(SP) = task id
8(SP) = notepad entry number
12(SP) = note value

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #NOTE_VALUE,-(SP)       * push note value
     move.l   #NOTE_NUM,-(SP)         * push entry number
     move.l   Task_id,-(SP)           * push task id
     move.l   #T_SETNOTE,D0           * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.4 INTERRUPT MANAGER

S.4.1 I_ENTER - Enter an ISR

INPUT:

D0 = function code

OUTPUT:

NONE

EXAMPLE:

```
        .
        .
        .

move.l   D0,-(SP)              * save task's D0
move.l   #I_ENTER,D0           * D0 = function code
jsr      rtems                 * enter the executive

* no need to check the return code here

        .
        .
        .
```

NOTES:

This directive uses the D0 register only. This register must be saved by the application before invoking I_ENTER. The D0 register is restored automatically by I_RETURN.

S.4.2 I_RETURN - Return from an ISR

INPUT:

D0 = function code

OUTPUT:

NONE

EXAMPLE:

```
          .
          .
          .

     move.l   #I_RETURN,D0          * D0 = function code
     jsr      rtems                 * enter the executive

     * will never return

          .
          .
          .
```

NOTES:

This directive uses only the D0 register. It restores D0 to its contents prior to I_ENTER.

This directive does not return to the caller.

S.5 TIME MANAGER

S.5.1 TM_SET - Set system date and time

INPUT:

D0 = function code
4(SP) = address of time_info data structure

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Time_struct,-(SP)      * push pointer to time buffer
     move.l   #TM_SET,D0              * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.5.2 TM_GET - Get system date and time

INPUT:

D0 = function code
4(SP) = address of time_info data structure

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Time_struct,-(SP)      * push pointer to time buffer
     move.l   #TM_GET,D0              * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.5.3 TM_WKAFTER - Wake up after interval

INPUT:

D0 = function code
4(SP) = length of interval (in ticks)

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

    move.l   #INTERVAL,-(SP)        * push ticks to wait
    move.l   #TM_WKAFTER,D0         * D0 = function code
    jsr      rtems                  * enter the executive

    * should check return code here

        .
        .
        .
```

S.5.4 TM_WKWHEN - Wake up when specified

INPUT:

D0 = function code
4(SP) = address of time_info data structure

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

    move.l   #Time_struct,-(SP)      * push time to wake
    move.l   #TM_WKWHEN,D0           * D0 = function code
    jsr      rtems                   * enter the executive

    * should check return code here

        .
        .
        .
```

S.5.5 TM_EVAFTER - Send event set after interval

INPUT:

D0 = function code
4(SP) = interval until event (in ticks)
8(SP) = event set
12(SP) = address of timer id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #Timer_id,-(SP)          * push pointer to timer id
move.l    #EVENTS,-(SP)            * push events to send
move.l    #INTERVAL,-(SP)          * push ticks until event
move.l    #TM_EVAFTER,D0           * D0 = function code
jsr       rtems                    * enter the executive

* should check return code here

          .
          .
          .
```

S.5.6 TM_EVWHEN - Send event set when specified

INPUT:

D0 = function code
4(SP) = address of time_info data structure
8(SP) = event set
12(SP) = address of timer id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Timer_id,-(SP)          * push pointer to timer id
move.l   #EVENTS,-(SP)            * push events to send
move.l   #Time_struct,-(SP)       * push time to send events
move.l   #TM_EVWHEN,D0            * D0 = function code
jsr      rtems                    * enter the executive

* should check return code here

        .
        .
        .
```

S.5.7 TM_EVEVERY - Send periodic event set

INPUT:

D0 = function code
4(SP) = interval between events (in ticks)
8(SP) = event set
12(SP) = address of timer id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Timer_id,-(SP)        * push pointer to timer id
     move.l   #EVENTS,-(SP)          * push events to send
     move.l   #INTERVAL,-(SP)        * push time between events
     move.l   #TM_EVEVERY,D0         * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.5.8 TM_CANCEL - Cancel timer event

INPUT:

D0 = function code
4(SP) = timer event id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l    Timer_id,-(SP)          * push timer id
     move.l    #TM_CANCEL,D0           * D0 = function code
     jsr       rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.5.9 TM_TICK - Announce a clock tick

INPUT:

D0 = function code

OUTPUT:

D0 = SUCCESSFUL

EXAMPLE:

```
          .
          .
          .

     move.l   #TM_TICK,D0          * D0 = function code
     jsr      rtems                * enter the executive

     * no need to check the return code here

          .
          .
          .
```

S.6 SEMAPHORE MANAGER

S.6.1 SM_CREATE - Create a semaphore

INPUT:

D0 = function code
4(SP) = user-defined four byte name
8(SP) = initial count
12(SP) = attributes
16(SP) = address of semaphore id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .
move.l   #Sem_id,-(SP)           * push pointer to semaphore id
move.l   #SEM_ATTRIBUTES,-(SP)   * push attributes
move.l   #INITIAL_COUNT,-(SP)    * push initial count
move.l   #SEM_NAME,-(SP)         * push name
move.l   #SM_CREATE,D0           * D0 = function code
jsr      rtems                   * enter the executive

* should check return code here

        .
        .
        .
```

S.6.2 SM_IDENT - Get ID of a semaphore

INPUT:

D0 = function code
4(SP) = user-defined name to search for
8(SP) = node identifier (defines search space)
12(SP) = address of semaphore id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Sem_id,-(SP)          * push pointer to semaphore id
     move.l   #NODE,-(SP)            * push node identifier
     move.l   #SEM_NAME,-(SP)        * push name
     move.l   #SM_IDENT,D0           * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.6.3 SM_DELETE - Delete a semaphore

INPUT:

D0 = function code
4(SP) = semaphore id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   Sem_id,-(SP)          * push semaphore id
     move.l   #SM_DELETE,D0         * D0 = function code
     jsr      rtems                 * enter the executive

     * should check return code here

          .
          .
          .
```

S.6.4 SM_P - Acquire a semaphore

INPUT:

D0 = function code
4(SP) = semaphore id
8(SP) = options
12(SP) = maximum interval to wait (in ticks)

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #INTERVAL,-(SP)        * push maximum ticks to wait
     move.l   #OPTIONS,-(SP)         * push options
     move.l   Sem_id,-(SP)           * push semaphore id
     move.l   #SM_P,D0               * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.6.5 SM_V - Release a semaphore

INPUT:

D0 = function code
4(SP) = semaphore id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    Sem_id,-(SP)          * push semaphore id
move.l    #SM_V,D0              * D0 = function code
jsr       rtems                 * enter the executive

* should check return code here

          .
          .
          .
```

S.7 MESSAGE MANAGER

S.7.1 Q_CREATE - Create a queue

INPUT:

D0 = function code
4(SP) = user-defined four byte name
8(SP) = maximum message count
12(SP) = attributes
16(SP) = address of queue id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Queue_id,-(SP)          * push pointer to queue id
move.l   #Q_ATTRIB,-(SP)          * push attributes
move.l   #MSG_BUF_COUNT,-(SP)     * push message count
move.l   #QUEUE_NAME,-(SP)        * push name
move.l   #Q_CREATE,D0             * D0 = function code
jsr      rtems                    * enter the executive

* should check return code here

        .
        .
        .
```

S.7.2 Q_IDENT - Get ID of a queue

INPUT:

D0 = function code
4(SP) = user-defined name to search for
8(SP) = node identifier (defines search space)
12(SP) = address of queue id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

    move.l   #Queue_id,-(SP)        * push pointer to queue id
    move.l   #NODE,-(SP)            * push node identifier
    move.l   #QUEUE_NAME,-(SP)      * push name
    move.l   #Q_IDENT,D0            * D0 = function code
    jsr      rtems                  * enter the executive

    * should check return code here

        .
        .
        .
```

S.7.3 Q_DELETE - Delete a queue

INPUT:

D0 = function code
4(SP) = queue id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   Queue_id,-(SP)          * push queue id
     move.l   #Q_DELETE,D0            * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.7.4 Q_SEND - Put message at rear of a queue

INPUT:

D0 = function code
4(SP) = queue id
8(SP) = address of message buffer

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Message,-(SP)          * push address of message
     move.l   Queue_id,-(SP)          * push queue id
     move.l   #Q_SEND,D0              * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.7.5 Q_URGENT - Put message at front of a queue

INPUT:

D0 = function code
4(SP) = queue id
8(SP) = address of message buffer

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

    move.l   #Message,-(SP)          * push address of message
    move.l   Queue_id,-(SP)          * push queue id
    move.l   #Q_URGENT,D0            * D0 = function code
    jsr      rtems                   * enter the executive

    * should check return code here

        .
        .
        .
```

S.7.6 Q_BROADCAST - Broadcast N messages to a queue

INPUT:

D0 = function code
4(SP) = queue id
8(SP) = address of message buffer
12(SP) = address of "number of tasks made ready" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Num_tasks,-(SP)        * push pointer to number
                                      *   of tasks readied
     move.l   #Message,-(SP)          * push address of message
     move.l   Queue_id,-(SP)          * push queue id
     move.l   #Q_BROADCAST,D0         * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.7.7 Q_RECEIVE - Receive message from a queue

INPUT:

D0 = function code
4(SP) = queue id
8(SP) = address of message buffer
12(SP) = options
16(SP) = maximum interval to wait (in ticks)

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #TIMEOUT,-(SP)          * push maximum ticks to wait
     move.l   #OPTIONS,-(SP)          * push receive options
     move.l   #Message,-(SP)          * push pointer to message
     move.l   Queue_id,-(SP)          * push queue id
     move.l   #Q_RECEIVE,D0           * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.7.8 Q_FLUSH - Flush all messages on a queue

INPUT:

D0 = function code
4(SP) = queue id
8(SP) = address of "number of messages flushed" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Num_flushed,-(SP)      * push pointer to number
                                      *   of messages flushed
     move.l   Queue_id,-(SP)          * push queue id
     move.l   #Q_FLUSH,D0             * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.8 EVENT MANAGER

S.8.1 EV_SEND - Send event set to a task

INPUT:

D0 = function code
4(SP) = task id to send events to
8(SP) = event set to send

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #EVENTS,-(SP)          * push events to send
     move.l   Task_id,-(SP)          * push task id
     move.l   #EV_SEND,D0            * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.8.2 EV_RECEIVE - Receive event condition

INPUT:

D0 = function code
4(SP) = input event condition
8(SP) = options
12(SP) = maximum interval to wait (in ticks)
16(SP) = address of events received storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Events_received,-(SP) * push pointer to events received
move.l   #TICKS,-(SP)           * push maximum ticks to wait
move.l   #OPTIONS,-(SP)         * push receive options
move.l   #EVENTS,-(SP)          * push event condition
move.l   #EV_RECEIVE,D0         * D0 = function code
jsr      rtems                  * enter the executive

* should check return code here

        .
        .
        .
```

S.9 SIGNAL MANAGER

S.9.1 AS_CATCH - Establish an ASR

INPUT:

D0 = function code
4(SP) = address of ASR
8(SP) = mode of ASR

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

    move.l   #ASR_MODE,-(SP)        * push ASR mode
    move.l   #Asr_handler,-(SP)     * push ASR address
    move.l   #AS_CATCH,D0           * D0 = function code
    jsr      rtems                  * enter the executive

    * should check return code here

        .
        .
        .
```

S.9.2 AS_SEND - Send signal set to a task

INPUT:

D0 = function code
4(SP) = task id
8(SP) = signal set

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

      move.l   #SIGNALS,-(SP)          * push signals to send
      move.l   Task_id,-(SP)           * push task id
      move.l   #AS_SEND,D0             * D0 = function code
      jsr      rtems                   * enter the executive

      * should check return code here

          .
          .
          .
```

S.9.3 AS_ENTER - Enter an ASR

INPUT:

D0 = function code

OUTPUT:

NONE

EXAMPLE:

```
          .
          .
          .

move.l    D0,-(SP)              * save task D0
move.l    #AS_ENTER,D0          * D0 = function code
jsr       rtems                 * enter the executive

* no need to check the return code here

          .
          .
          .
```

NOTES:

This directive uses the D0 register only. This register must be saved by the application before invoking AS_ENTER. The D0 register is restored automatically by AS_RETURN.

S.9.4 AS_RETURN - Return from an ASR

INPUT:

D0 = function code

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #AS_RETURN,D0          * D0 = function code
     jsr      rtems                  * enter the executive

     * does not return if SUCCESSFUL

          .
          .
          .
```

NOTES:

This directive uses only the D0 register. It restores D0 to its contents prior to AS_ENTER.

This directive does not return to the caller.

S.10 PARTITION MANAGER

S.10.1 PT_CREATE - Create a partition

INPUT:

D0 = function code
4(SP) = user-defined four byte name
8(SP) = physical start address of partition
12(SP) = length (in bytes)
16(SP) = size of buffers (in bytes)
20(SP) = attributes
24(SP) = address of partition id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Part_id,-(SP)          * push pointer to partition id
move.l   #PART_ATTRIBUTES,-(SP)  * push attributes
move.l   #BUF_SIZE,-(SP)         * push buffer size
move.l   #PART_LENGTH,-(SP)      * push length
move.l   #PART_ADDR,-(SP)        * push start address
move.l   #PART_NAME,-(SP)        * push name
move.l   #PT_CREATE,D0           * D0 = function code
jsr      rtems                   * enter the executive

* should check return code here

        .
        .
        .
```

S.10.2 PT_IDENT - Get ID of a partition

INPUT:

D0 = function code
4(SP) = user-defined name to search for
8(SP) = node identifier (defines search space)
12(SP) = address of partition id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Part_id,-(SP)          * push pointer to partition id
move.l   #NODE,-(SP)             * push node identifier
move.l   #PART_NAME,-(SP)        * push name
move.l   #PT_IDENT,D0            * D0 = function code
jsr      rtems                   * enter the executive

* should check return code here

        .
        .
        .
```

S.10.3 PT_DELETE - Delete a partition

INPUT:

D0 = function code
4(SP) = partition id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   Part_id,-(SP)          * push partition id
     move.l   #PT_DELETE,D0          * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.10.4 PT_GETBUF - Get buffer from a partition

INPUT:

D0 = function code
4(SP) = partition id
8(SP) = address of "buffer address" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Buff_addr,-(SP)        * push pointer to buffer address
     move.l   Part_id,-(SP)           * push partition id
     move.l   #PT_GETBUF,D0           * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.10.5 PT_RETBUF - Return buffer to a partition

INPUT:

D0 = function code
4(SP) = partition id
8(SP) = buffer address

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   Buff_addr,-(SP)        * push buffer address
     move.l   Part_id,-(SP)          * push partition id
     move.l   #PT_RETBUF,D0          * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.11 REGION MANAGER

S.11.1 RN_CREATE - Create a region

INPUT:

D0 = function code
4(SP) = user-defined four byte name
8(SP) = physical start address of region
12(SP) = length (in bytes)
16(SP) = page size (in bytes)
20(SP) = attributes
24(SP) = address of region id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Regn_id,-(SP)          * push pointer to region id
     move.l   #REGN_ATTRIB,-(SP)      * push attributes
     move.l   #REGN_PAGE,-(SP)        * push page size
     move.l   #REGN_LENGTH,-(SP)      * push length
     move.l   #REGN_ADDRESS,-(SP)     * push starting address
     move.l   #REGN_NAME,-(SP)        * push name
     move.l   #RN_CREATE,D0           * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.11.2 RN_IDENT - Get ID of a region

INPUT:

D0 = function code
4(SP) = user-defined name to search for
8(SP) = address of region id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Regn_id,-(SP)          * push pointer to region id
     move.l   #REGN_NAME,-(SP)        * push name
     move.l   #RN_IDENT,D0            * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.11.3 RN_DELETE - Delete a region

INPUT:

D0 = function code
4(SP) = region id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   Regn_id,-(SP)          * push region id
move.l   #RN_DELETE,D0          * D0 = function code
jsr      rtems                  * enter the executive

* should check return code here

        .
        .
        .
```

S.11.4 RN_GETSEG - Get segment from a region

INPUT:

D0 = function code
4(SP) = region id
8(SP) = segment size desired (in bytes)
12(SP) = options
16(SP) = maximum interval to wait (in ticks)
20(SP) = address of "segment address" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Seg_addr,-(SP)        * push pointer to segment address
move.l   #TIMEOUT,-(SP)         * push maximum ticks to wait
move.l   #OPTIONS,-(SP)         * push getseg options
move.l   #SEG_SIZE,-(SP)        * push desired segment size
move.l   Regn_id,-(SP)          * push region id
move.l   #RN_GETSEG,D0          * D0 = function code
jsr      rtems                  * enter the executive

* should check return code here

        .
        .
        .
```

S.11.5 RN_RETSEG - Return segment to a region

INPUT:

D0 = function code
4(SP) = region id
8(SP) = segment address

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   Seg_addr,-(SP)          * push segment address
     move.l   Regn_id,-(SP)           * push region id
     move.l   #RN_RETSEG,D0           * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.12 DUAL-PORTED MEMORY MANAGER

S.12.1 DP_CREATE - Create a port

INPUT:

D0 = function code
4(SP) = user-defined four byte name
8(SP) = starting internal address
12(SP) = starting external address
16(SP) = length (in bytes)
20(SP) = address of port id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
        .
        .
        .

move.l   #Port_id,-(SP)          * push pointer to port id
move.l   #PORT_LENGTH,-(SP)      * push length of DPMA
move.l   External_addr,-(SP)     * push external address
move.l   Internal_addr,-(SP)     * push internal address
move.l   #PORT_NAME,-(SP)        * push name
move.l   #DP_CREATE,D0           * D0 = function code
jsr      rtems                   * enter the executive

* should check return code here

        .
        .
        .
```

S.12.2 DP_IDENT - Get ID of a port

INPUT:

D0 = function code
4(SP) = user-defined name to search for
8(SP) = address of port id storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Port_id,-(SP)          * push pointer to port id
     move.l   #PORT_NAME,-(SP)        * push name
     move.l   #DP_IDENT,D0            * D0 = function code
     jsr      rtems                   * enter the executive

     * should check return code here

          .
          .
          .
```

S.12.3 DP_DELETE - Delete a port

INPUT:

D0 = function code
4(SP) = port id

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   Port_id,-(SP)          * push port id
     move.l   #DP_DELETE,D0          * D0 = function code
     jsr      rtems                  * enter the executive

     * should check return code here

          .
          .
          .
```

S.12.4 DP_2INTERNAL - Convert external to internal address

INPUT:

D0 = function code
4(SP) = port id
8(SP) = external address
12(SP) = address of "internal address" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #Internal_addr,-(SP)    * push pointer to internal
address
move.l    External_addr,-(SP)     * push external address
move.l    Port_id,-(SP)           * push port id
move.l    #DP_2INTERNAL,D0        * D0 = function code
jsr       rtems                   * enter the executive

* should check return code here

          .
          .
          .
```

S.12.5 DP_2EXTERNAL - Convert internal to external address

INPUT:

D0 = function code
4(SP) = port id
8(SP) = internal address
12(SP) = address of "external address" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #External_addr,-(SP)    * push pointer to external
address
move.l    Internal_addr,-(SP)     * push internal address
move.l    Port_id,-(SP)           * push port id
move.l    #DP_2EXTERNAL,D0        * D0 = function code
jsr       rtems                   * enter the executive

* should check return code here

          .
          .
          .
```

S.13 INPUT/OUTPUT MANAGER

S.13.1 DE_INIT - Initialize a device driver

INPUT:

D0 = function code
4(SP) = device number
8(SP) = address of parameter block
12(SP) = address of "return code from device driver" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #Drv_rval,-(SP)        * push pointer to driver's
                                 *   return code
move.l    #Param_block,-(SP)     * push pointer to parameter block
move.l    #DEV_NUM,-(SP)         * push device number
move.l    #DE_INIT,D0            * D0 = function code
jsr       rtems                  * enter the executive

* should check directive's return code here
* should check driver's return code here

          .
          .
          .
```

S.13.2 DE_OPEN - Open a device

INPUT:

D0 = function code
4(SP) = device number
8(SP) = address of parameter block
12(SP) = address of "return code from device driver" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #Drv_rval,-(SP)        * push pointer to driver's
                                 *   return code
move.l    #Param_block,-(SP)     * push pointer to parameter block
move.l    #DEV_NUM,-(SP)         * push device number
move.l    #DE_OPEN,D0            * D0 = function code
jsr       rtems                  * enter the executive

* should check directive's return code here
* should check driver's return code here

          .
          .
          .
```

S.13.3 DE_CLOSE - Close a device

INPUT:

D0 = function code
4(SP) = device number
8(SP) = address of parameter block
12(SP) = address of "return code from device driver" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #Drv_rval,-(SP)        * push pointer to driver's
                                 *   return code
move.l    #Param_block,-(SP)     * push pointer to parameter block
move.l    #DEV_NUM,-(SP)         * push device number
move.l    #DE_CLOSE,D0           * D0 = function code
jsr       rtems                  * enter the executive

* should check directive's return code here
* should check driver's return code here

          .
          .
          .
```

S.13.4 DE_READ - Read from a device

INPUT:

D0 = function code
4(SP) = device number
8(SP) = address of parameter block
12(SP) = address of "return code from device driver" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #Drv_rval,-(SP)          * push pointer to driver's
                                   *   return code
move.l    #Param_block,-(SP)       * push pointer to parameter block
move.l    #DEV_NUM,-(SP)           * push device number
move.l    #DE_READ,D0              * D0 = function code
jsr       rtems                    * enter the executive

* should check directive's return code here
* should check driver's return code here

          .
          .
          .
```

S.13.5 DE_WRITE - Write to a device

INPUT:

D0 = function code
4(SP) = device number
8(SP) = address of parameter block
12(SP) = address of "return code from device driver" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

move.l    #Drv_rval,-(SP)        * push pointer to driver's
                                 *   return code
move.l    #Param_block,-(SP)     * push pointer to parameter block
move.l    #DEV_NUM,-(SP)         * push device number
move.l    #DE_WRITE,D0           * D0 = function code
jsr       rtems                  * enter the executive

* should check directive's return code here
* should check driver's return code here

          .
          .
          .
```

S.13.6 DE_CNTRL - Special device services

INPUT:

D0 = function code
4(SP) = device number
8(SP) = address of parameter block
12(SP) = address of "return code from device driver" storage location

OUTPUT:

D0 = directive status code

EXAMPLE:

```
          .
          .
          .

     move.l   #Drv_rval,-(SP)        * push pointer to driver's
                                     *   return code
     move.l   #Param_block,-(SP)     * push pointer to parameter block
     move.l   #DEV_NUM,-(SP)         * push device number
     move.l   #DE_CNTRL,D0           * D0 = function code
     jsr      rtems                  * enter the executive

     * should check directive's return code here
     * should check driver's return code here

          .
          .
          .
```

S.14 FATAL ERROR MANAGER

S.14.1 K_FATAL - Invoke the fatal error handler

INPUT:

D0 = function code
4(SP) = error code

OUTPUT:

NONE

EXAMPLE:

```
          .
          .
          .

move.l    Fatal_error,-(SP)      * push error code
move.l    #K_FATAL,D0            * D0 = function code
jsr       rtems                  * enter the executive

* will never return

          .
          .
          .
```

NOTES:

This directive does not return to the caller.

S.15 MULTIPROCESSING

S.15.1 MP_ANNOUNCE - Announce the arrival of a packet

INPUT:

D0 = function code

OUTPUT:

NONE

EXAMPLE:

```
          .
          .
          .

     move.l   #MP_ANNOUNCE,D0        * D0 = function code
     jsr      rtems                  * enter the executive

     * no need to check the return code here

          .
          .
          .
```

S.16 directives.eq

```
************************************************************
*
*  directives.eq
*
*  The following definitions are the directive numbers
*  used in the assembly interface.
*
*  NOTE: For standard Motorola MC680x0 series assemblers.
*

INIT_EXEC           EQU  0
I_ENTER             EQU  1
I_RETURN            EQU  2
K_FATAL             EQU  3
TM_SET              EQU  4
TM_GET              EQU  5
TM_WKAFTER          EQU  6
TM_WKWHEN           EQU  7
TM_EVAFTER          EQU  8
TM_EVWHEN           EQU  9
TM_EVEVERY          EQU 10
TM_CANCEL           EQU 11
TM_TICK             EQU 12
T_CREATE            EQU 13
T_IDENT             EQU 14
T_START             EQU 15
T_RESTART           EQU 16
T_DELETE            EQU 17
T_SUSPEND           EQU 18
T_RESUME            EQU 19
T_SETPRI            EQU 20
T_MODE              EQU 21
T_GETNOTE           EQU 22
T_SETNOTE           EQU 23
EV_SEND             EQU 24
EV_RECEIVE          EQU 25
AS_CATCH            EQU 26
AS_SEND             EQU 27
AS_ENTER            EQU 28
AS_RETURN           EQU 29
Q_CREATE            EQU 30
Q_IDENT             EQU 31
Q_DELETE            EQU 32
Q_SEND              EQU 33
Q_URGENT            EQU 34
```

```
Q_BROADCAST             EQU 35
Q_RECEIVE               EQU 36
Q_FLUSH                 EQU 37
SM_CREATE               EQU 38
SM_IDENT                EQU 39
SM_DELETE               EQU 40
SM_P                    EQU 41
SM_V                    EQU 42
RN_CREATE               EQU 43
RN_IDENT                EQU 44
RN_DELETE               EQU 45
RN_GETSEG               EQU 46
RN_RETSEG               EQU 47
PT_CREATE               EQU 48
PT_IDENT                EQU 49
PT_DELETE               EQU 50
PT_GETBUF               EQU 51
PT_RETBUF               EQU 52
DP_CREATE               EQU 53
DP_IDENT                EQU 54
DP_DELETE               EQU 55
DP_2INTERNAL            EQU 56
DP_2EXTERNAL            EQU 57
MP_ANNOUNCE             EQU 58
DE_INIT                 EQU 59
DE_OPEN                 EQU 60
DE_CLOSE                EQU 61
DE_READ                 EQU 62
DE_WRITE                EQU 63
DE_CNTRL                EQU 64

BEGIN_CODE_DCL
  global rtems              * single RTEMS entry point
END_CODE_DCL

*  end of directives.eq
*
*************************************************************
```

S.17 dirstatus.eq

```
*************************************************************
*
*  dirstatus.eq
*
*  This include file contains the status codes returned
*  from the executive's directives.
*
*  NOTE: For standard Motorola MC680x0 series assemblers.
*

SUCCESSFUL       EQU  0    * successful completion
E_EXITTED        EQU  1    * returned from a task
E_NOMP           EQU  2    * multiprocessing not configured
E_NAME           EQU  3    * invalid object name
E_ID             EQU  4    * invalid object id
E_TOOMANY        EQU  5    * too many
E_TIMEOUT        EQU  6    * timed out waiting
E_DELETE         EQU  7    * object was deleted while waiting
E_SIZE           EQU  8    * specified size was invalid
E_ADDRESS        EQU  9    * address specified is invalid
E_NUMBER         EQU 10    * number was invalid
E_NOTDEFINED     EQU 11    * item has been initialized
E_INUSE          EQU 12    * resources still outstanding
E_UNSATISFIED    EQU 13    * request not satisfied
E_STATE          EQU 14    * task is in wrong state
E_ALREADY        EQU 15    * task already in state
E_SELF           EQU 16    * illegal on calling task
E_REMOTE         EQU 17    * illegal on remote object
E_CALLED         EQU 18    * called from wrong environment
E_PRIORITY       EQU 19    * invalid task priority
E_CLOCK          EQU 20    * invalid date/time
E_NODE           EQU 21    * invalid node id
E_NOTCONFIGURED  EQU 22    * directive not configured
E_NOTIMPLEMENTED EQU 23    * directive not implemented

* end of dirstatus.eq
*
*************************************************************
```

INITIAL DISTRIBUTION

	Copies
U.S. Army Materiel System Analysis Activity ATTN: AMXSY-MP (Herbert Cohen) Aberdeen Proving Ground, MD 21005	1
IIT Research Institute ATTN: GACIAC 10 W. 35th Street Chicago, IL 60616	1
WL/MNAG ATTN: Chris Anderson Eglin AFB, FL 32542-5434	1
Naval Weapons Center Missile Software Technology Office Code 3901C, ATTN: Mr. Carl W. Hall China Lake, CA 93555-6001	1
On-line Applications Research 3315 Memorial Parkway, SW Huntsville, AL 35801	3
CEA Incorporated Blue Hills Office Park 150 Royall Street Suite 260, ATTN: Mr. John Shockro Canton, MA 01021	1
VITA 10229 N. Scottsdale Rd Suite B, ATTN: Mr. Ray Alderman Scottsdale, AZ 85253	1
Westinghouse Electric Corp. P.O. Box 746 - MS432 ATTN: Mr. Eli Solomon Baltimore, MD 21203	1
Dept. of Computer Science B-173 Florida State University ATTN: Dr. Ted Baker Tallahassee, FL 32306-4019	1
DSD Laboratories 75 Union Avenue ATTN: Mr. Roger Whitehead Studbury, MA 01776	1

	Copies
AMSMI-RD	1
AMSMI-RD-GS, Dr. Paul Jacobs	1
AMSMI-RD-GC-S, Gerald E. Scheiman	1
Wanda M. Hughes	5
Phillip Acuff	4
AMSMI-RD-BA	1
AMSMI-RD-BA-C3, Bob Christian	1
AMSMI-RD-SS	1
AMSMI-RD-CS-R	15
AMSMI-RD-CS-T	1
AMSMI-GC-IP, Mr. Fred M. Bush	1
CSSD-CR-S, Mr. Frank Poslajko	1
SFAE-FS-ML-TM, Mr. Frank Gregory	1
SFAE-AD-ATA-SE, Mr. Julian Cothran	1
Mr. John Carter	1

☆ U. S. GOVERNMENT PRINTING OFFICE 1992 631-100 / 60224